NL030436

Europäisches Patentamt | European Patent Office | Office européen des brevets

IB/2004/50422

Bescheinigung | Certificate | Attestation

Die angehefteten Unterlagen stimmen mit der ursprünglich eingereichten Fassung der auf dem nächsten Blatt bezeichneten europäischen Patentanmeldung überein.

The attached documents are exact copies of the European patent application described on the following page, as originally filed.

Les documents fixés à cette attestation sont conformes à la version initialement déposée de la demande de brevet européen spécifiée à la page suivante.

Patentanmeldung Nr. | Patent application No. | Demande de brevet n°

03076102.7

PRIORITY DOCUMENT
SUBMITTED OR TRANSMITTED IN COMPLIANCE WITH RULE 17.1(a) OR (b)

Der Präsident des Europäischen Patentamts;
Im Auftrag

For the President of the European Patent Office

Le Président de l'Office européen des brevets
p.o.

R C van Dijk

Anmeldung Nr:
Application no.: 03076102.7
Demande no:

Anmeldetag:
Date of filing: 15.04.03
Date de dépôt:

Anmelder/Applicant(s)/Demandeur(s):

Koninklijke Philips Electronics N.V.
Groenewoudseweg 1
5621 BA Eindhoven
PAYS-BAS

Bezeichnung der Erfindung/Title of the invention/Titre de l'invention:
(Falls die Bezeichnung der Erfindung nicht angegeben ist, siehe Beschreibung.
If no title is shown please refer to the description.
Si aucun titre n'est indiqué se referer à la description.)

Computer graphics processor and method for generating a computer graphics image

In Anspruch genommene Priorität(en) / Priority(ies) claimed /Priorité(s) revendiquée(s)
Staat/Tag/Aktenzeichen/State/Date/File no./Pays/Date/Numéro de dépôt:

Internationale Patentklassifikation/International Patent Classification/
Classification internationale des brevets:

G06T/

Am Anmeldetag benannte Vertragstaaten/Contracting states designated at date of filing/Etats contractants désignées lors du dépôt:

AT BE BG CH CY CZ DE DK EE ES FI FR GB GR HU IE IT LU MC NL
PT RO SE SI SK TR LI

Computer graphics processor and method for generating a computer graphics image

The invention relates to a computer graphics processor.
The invention further relates to a method for generating a computer graphics image.

It is known that 2D filtering can be approximated using two 1D filter passes for a large class of filters, providing more efficient solutions. See for example: Edwin Catmull and Alvy Ray Smith. 3-d transformations of images in scanline order. In *Computer Graphics (SIGGRAPH '80 Proceedings)*, volume 14(3), pages 279 – 285, 1980 and George Wolberg and Terrance E. Boult. Separable image warping with spatial lookup tables. In *ComputerGraphics (Proc. Siggraph '89)*, volume 23(3), pages 369 – 378, July 1989.
As described therein, such separation can exhibit so-called bottleneck and shear problems, resulting in blurriness and aliasing respectively. In order to moderate the consequence of these disadvantageous effects these publications describe four traversal orders. One of these is chosen to reduce the bottleneck problem. As a solution to shear aliasing, supersampling is described in the known literature. Supersampling requires an extra downscale filter to reduce the extra resolution to the required output resolution.

It is a purpose of the present invention to provide a computer graphics processor and method for generating a computer graphics image wherein such an extra downscale filter is superfluous. A computer graphics processor of the invention in accordance with this purpose is claimed in claim 1. A method for generating a computer graphics image of the invention in accordance with this purpose is claimed in claim 2. In the computer graphics processor according to the invention the model information providing unit provides information representing a set of graphics primitives, such as triangles or other polygons, or Beziershapes. The information provided may comprise geometrical information defining a shape of the primitives and/or appearance information defining an appearance of the primitives, such as texture and color information.
The rasterizer is capable of generating a first sequence of coordinates which coincide with a base grid u,v associated with the primitive, and is further capable of generating one or more sequences of interpolated values associated with the first sequence. The further sequence may include screen (display) coordinates x,y and for example includes normal information for the surface represented by the primitive.
The color generator is arranged for assigning a color to said first sequence of coordinates using said appearance information. The color generator may simply use an interpolated color provided by the rasterizer, but may also perform complicated shading and texturing operations. The display space resampler is arranged for resampling the color assigned by the color generator in the base grid for coordinates u,v to a representation in a grid associated with a display with coordinates x,y. The transformation is carried out in a first and a second transformation as this substantialy reduces computational effort.
The display space resampler uses a transposing facility and a selection facility for selecting from four options depending on whether the image data is transposed or not, and depending on the order wherein the first and the second transformation are applied, i.e. whether first the x or the y coordinate of the display coordinates is calculated. This computer graphics

processor and the method according to the invention include a new selection criterium to choose between the four alternatives, which also takes shear into account. It was recognized by the inventors that the second pass filter can act as downsample filter for any supersampling performed in the first pass, while supersampling in the second pass would requires an extra downsample filter (a third pass). Therefore, after a first selection step to reduce bottleneck, wherein it is determined whether a transposition should be applied or not, a choice between the remaining two options is made on the shear occurring in the two passes (transformations).

A transposition is intended to include transformations wherein the coordinates in a coordinate pair are interchanged. This is for example achieved by mirroring the coordinates around a line x=y or a line x=-y. The same effect is achieved by a forward or a backward rotation over 90°. In this case however, the second transposition should involve forward rotation if the first tranposition is a backward rotation and vice versa.

The selected option is the one wherein the least amount of shear occurs in the second pass, so that supersampling in the second pass is avoided. Instead, the worst shear is put in the first pass where shear aliasing can easily be reduced by using supersampling.

More in particular four options can be defined as follows.

- a first option where:
  in a first pass, uv image data is resampled into xv image data by mapping u to x for each line having coordinate v, and
  in a second pass the xv image data is transformed into xy image data by mapping v to y for each line having coordinate x,
- a second option where
  in a first pass, uv image data is resampled into uy image data by mapping v to y for each line having coordinate u, and
  in a second pass the uy image data is transformed into xy image data by mapping u to x for each line having coordinate y,
- a third option where
  in a first pass, uv image data is transformed into yv image data by mapping u to y for each line having coordinate v,
  in a second pass the yv image data is transformed into yx image data by mapping v to x for each line having coordinate y, and
  the yx image data is transposed to xy image data,
- a fourth option where
  in a first pass, uv image data is transformed into ux image data by mapping v to x for each line having coordinate u,
  in a second pass the ux image data is transformed into yx image data by mapping u to y for each line having coordinate x, and
  the yx image data is transposed to xy image data.

In the third and the fourth option an explicit transposition of the coordinates generated by the rasterizer is not necessary. Instead the coordinates of the vertices and/or control points defining the primitives can simply be transposed before generating the coordinates of the base grid.

These and other aspects of the invention are described in more detail in the attached article **"Pixel shading and forward texture mapping"**, pp 1-7, by Bart Barenbrug, intended for publication in Graphics Hardware (2003), M. Doggett, W. Heidrich, B. Mark, A. Schilling (Editors).

Possible implementations of the model information providing unit, the rasterizer and the color generator are described in the European patent application 03100313.0 filed 13FEB2003 by the same inventors.

CLAIMS:

1. A computer graphics processor comprising a model information providing unit for providing information representing a set of graphics primitives, a rasterizer capable of generating a first sequence of coordinates which coincide with a base grid associated with the primitive, a color generator for assigning a color to said first sequence of coordinates, and a display space resampler for resampling the color assigned by the color generator in the base grid for coordinates u,v to a representation in a grid associated with a display with coordinates x,y, in a first and a second transformation, carried out in a first and a second pass, and optionally including a transposition,
the processor having a selection facility for selecting the order of the transformations and selecting whether to apply a transposition or not based on an evaluation of the partial derivatives
$\frac{\partial x}{\partial u}, \frac{\partial x}{\partial v}, \frac{\partial y}{\partial u}, \frac{\partial y}{\partial v}$, two of which determine shear and two of which determine scaling in the transformations, the selection being made wherein relatively large derivatives occur as scale factors, and relatively small derivatives occur as shear factors and wherein the lowest amount of shear occurs during the second transformation.

2. Method for generating a computer graphics image, comprising the following steps:
providing information representing a set of graphics primitives,
generating a first sequence of coordinates which coincide with a base grid associated with the primitive,
assigning a color to said first sequence of coordinates using information representing the graphics primitives,
resampling the color assigned by the color generator in the base grid for coordinates u,v to a representation in a grid associated with a display with coordinates x,y, in a first and a second pass,

evaluating the partial derivatives $\frac{\partial x}{\partial u}, \frac{\partial x}{\partial v}, \frac{\partial y}{\partial u}, \frac{\partial y}{\partial v}$ of the coordinates in the display with respect to the coordinates in the base grid, two of which determine shear and two of which determine scaling in the transformations, the selection being made wherein relatively large derivatives occur as scale factors, and relatively small derivatives occur as shear factors and wherein the lowest amount of shear occurs during the second transformation.

3. Computer graphics processor according to claim 1 or 2, where in the execution of the third and the fourth option the second transposition and the second transformation are combined.

4. Method for generating a computer graphics image, comprising the following steps:

- providing information representing a set of graphics primitives, the information comprising at least geometrical information defining a shape of the primitives and appearance information defining an appearance of the primitives,
- generating a first sequence of coordinates which coincide with a base grid associated with the primitive, and generating one or more sequences of interpolated values associated with the first sequence,
- assigning a color to said first sequence of coordinates using said appearance information,
- resampling the color assigned by the color generator in the base grid for coordinates u,v to a representation in a grid associated with a display with coordinates x,y, in a first and a second pass
- evaluating the partial derivatives $\frac{\partial x}{\partial u}, \frac{\partial x}{\partial v}, \frac{\partial y}{\partial u}, \frac{\partial y}{\partial v}$ of the coordinates in the display with respect to the coordinates in the base grid,

-selecting one of the following options:

- a first option where:

in a first pass, uv image data is resampled into xv image data by mapping u to x for each line having coordinate v, and

- in a second pass the xv image data is transformed into xy image data by mapping v to y for each line having coordinate x,

a second option where
in a first pass, uv image data is resampled into uy image data by mapping v to y for each line having coordinate u, and
in a second pass the uy image data is transformed into xy image data by mapping u to x for each line having coordinate y,
a third option where
in a first pass, uv image data is transformed into yv image data by mapping u to y for each line having coordinate v,
in a second pass the yv image data is transformed into yx image data by mapping v to x for each line having coordinate y, and
the yx image data is transposed to xy image data,
a fourth option where
in a first pass, uv image data is transformed into ux image data by mapping v to x for each line having coordinate u,
in a second pass the ux image data is transformed into yx image data by mapping u to y for each line having coordinate x, and
the yx image data is transposed to xy image data,

from the options to transpose or not the option being selected where a matrix representing the result of the first and the second filter has relatively high scale factors and the order for appling the first filter and the second filter being selected, wherein the highest shear occurs during the first pass.

ABSTRACT:

A computer graphics processor is described comprising a model information providing unit for providing information representing a set of graphics primitives, a rasterizer capable of generating a first sequence of coordinates which coincide with a base grid associated with the primitive, a color generator for assigning a color to said first sequence of coordinates, and a display space resampler for resampling the color assigned by the color generator in the base grid for coordinates u,v to a representation in a grid associated with a display with coordinates x,y, in a first and a second transformation. The transformation is carried out in a first and a second pass, and optionally includes a transposition. The order of the passes and the decision to apply a transposition or not is based on an evaluation of the partial derivatives $\frac{\partial x}{\partial u}, \frac{\partial x}{\partial v}, \frac{\partial y}{\partial u}, \frac{\partial y}{\partial v}$, two of which determine shear and two of which determine scaling in the transformations. The option is selected wherein relatively large derivatives occur as scale factors, and relatively small derivatives occur as shear factors and wherein the lowest amount of shear occurs during the second transformation.

Figure 3 of pdf article.

Graphics Hardware (2003), pp. 1–8
M. Doggett, W. Heidrich, B. Mark, A. Schilling (Editors)

Pixel shading and forward texture mapping

Bart Barenbrug

Philips Research Laboratories Eindhoven, Prof. Holstlaan 4, 5656 AA Eindhoven, The Netherlands
Bart.Barenbrug@philips.com

Abstract

Programmable vertex and pixel shading has in recent years allowed for real-time colour computation providing very realistic appearances to rendered objects. During the rendering process, another aspect which contributes considerably to the fidelity of the final pictures is anti-aliasing. As super-sampling and multi-sampling approaches are driven towards more and more subsamples per pixel to increase quality, pre-filtering approaches become more economical. Earlier work shows that forward texture mapping allows such pre-filtering at a costs comparable to 2x2 super-sampling, while delivering a quality-level on par with 4x4 supersampling. But architectures proposed for this technique so far have lacked programmable pixel shading capabilities. This paper presents an architecture where pixel shaders are used to compute colours across the surface of a primitive using traditional programmable pixel shading techniques, and where afterwards the resulting "texture map" is mapped onto the screen using forward texture mapping. This combined architecture features both the advantages provided by programmable pixel shading and the high quality anti-aliasing provided by the prefilter technique.

Categories and Subject Descriptors (according to ACM CCS): I.3.1 [Computer Graphics]: Graphics processors I.3.3 [Computer Graphics]: Display algorithms I.3.7 [Computer Graphics]: Color, shading, shadowing, and texture

1. Introduction

Programmable vertex and pixel shading has in recent years paved the way towards more programmable graphics pipeline architectures and opened up a host of possibilities for improving the fidelity of real-time computer generated imagery, enabling effects similar to those used in off-line rendering for the movie industry.

Another issue when it comes to high-fidelity pictures is anti-aliasing (both texture anti-aliasing and edge anti-aliasing). Current super-sampling and multi-sampling techniques are computationally intensive and require a lot of memory bandwidth. In a graphics pipeline architecture using forward texture mapping, prefiltering can easily be used to avoid aliasing artifacts[8,9,2]. Such systems so far however do not support pixel shading.

This paper presents a novel graphics pipeline architecture featuring both programmable shaders and forward texture mapping prefiltering techniques, providing both high-quality anti-aliasing and all the pixel shading effects. The changes required to transform a traditional pipeline into this new architecture are presented, along with details on for example how to handle mipmapping in a forward mapping architecture, and on how to solve the bottleneck and shear problems that occur when using two-pass resampling.

The paper is structured as follows: in Section 2, the previous work will be presented in the form of a walk down the traditional pixel shading pipeline and the forward texture mapping pipeline. Then, in Section 3, the combined architecture will be presented and some of its features discussed. Results are then presented in Section 4 followed by some conclusions in Section 5.

2. Previous Work

Rendering computer generated images is done using many different techniques for many different purposes. High-volume consumer level products are based on screen-space scanline algorithms which render primitives one by one by traversing the pixels on the screen to which the primitive projects. Inverse texture mapping is used to determine the colour that a primitive contributes to such a pixel.

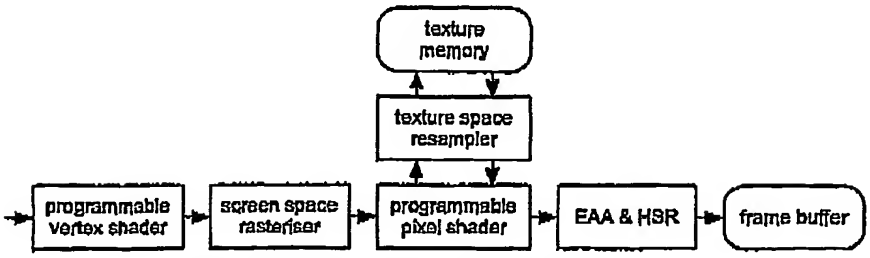


Figure 1: *A traditional pixel shading architecture*

With some modifications[8] in the later stages (rasterisation and beyond) of such a graphics pipeline, forward texture mapping can be used as well. With forward texture mapping, the projection of the primitive on the texture map is traversed, rather than the projection on the screen, and the texture samples (texels) which are encountered are mapped and splatted onto the screen. The sometimes quite inverse nature of programmable pixel shading (used when for example doing dependent texture look-ups, or performing environment-mapped bump mapping) does not seem to fit well with this technique. Forward texture mapping does however provide very high-quality anti-aliasing at moderate cost.

Before presenting a combined architecture, first the traditional pixel shading architecture is discussed to sketch the context, as well as the forward texture mapping architecture described in the paper by Meinds[8]. The building blocks from these two architectures can then later be combined into an architecture featuring both techniques.

2.1. Traditional pixel shading architectures

Figure 1 shows the architecture of the last stages of a graphics pipeline featuring programmable vertex and pixel shading.

The programmable vertex shader (together with the frame buffer at the end of the pipeline) provides the context of the changes to the architecture which we will describe later. The vertex shader[6] can modify and setup data for the pixel shader for each vertex of a primitive that is to be rendered. The data provided by the vertex shader to the rasteriser (for interpolation) usually includes attributes like diffuse and/or specular colour, texture coordinates, (homogeneous) screen coordinates, and sometimes extra data like surface normals or other data required for the shading process.

These attributes are offered to the screen-space rasteriser which uses a scanline algorithm to traverse the pixels which lie within the projection of the primitive on the screen, by selecting the screen coordinates from the vertex attributes as driving variables for the rasterisation process. The rasteriser interpolates the attributes to provide values for each of the pixels visited. Interpolation accounts for the perspective mapping from world space to screen space.

The attributes are then available at each pixel location, where a pixel shader[11, 12, 13] can use them to compute the local surface colour. When texture data is needed, the texture space resampler is used to obtain a texture sample given the texture coordinates. These texture coordinates are generated by the pixel shader based on the interpolated coordinates received from the rasteriser and any results from previous texture fetches (so-called dependent texturing) and/or calculations. The texture filter operation is usually based on bi-linear or tri-linear interpolation of nearby texels, or combinations of such texture probes to approximate an anisotropic (perspectively transformed) filter footprint.

After the surface colour for a pixel has been determined, the resulting pixel fragment is sent onwards to the Edge Anti-Aliasing and Hidden Surface Removal (EAA & HSR) unit. Usually, this unit uses a Z-buffer for hidden surface removal and multi-sampling (with the associated sub-sample buffer and downsample logic) for edge anti-aliasing. Downsampling is applied when all primitives have been rendered using a (usually box) prefilter to combine sub-samples to colours at the final pixel resolution.

2.2. A forward texture mapping architecture

A forward texture mapping architecture, further described in the paper by Meinds[8], is depicted in Figure 2. This architecture does not support pixel shading, and the vertex shader is a traditional Transform and Lighting unit.

As in the traditional pipeline, the T&L unit delivers attributes for the vertices of a primitive. Unlike the traditional screen space rasteriser, the texture space rasteriser traverses the projection of the primitive onto the texture map (rather than the screen), by selecting the texture coordinates (instead of screen coordinates) as the driving variables for the rasterisation process. All attributes are interpolated (linearly, except for the screen coordinates to which a texel is projected, which are interpolated perspectively).

The texture space rasteriser traverses the texture map on a

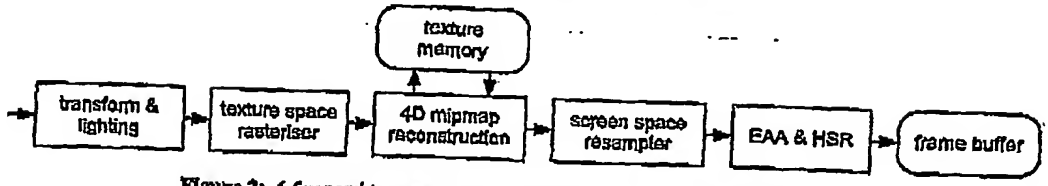


Figure 2: *A forward texture mapping architecture without pixel shading*

grid corresponding to 4D mipmapping (for more details on 4D mipmaps, see Heckbert's survey of texture mapping[4]). A texture fetch then amounts to 4D mipmap reconstruction from the 3D mipmap data stored in the texture memory. A fetched texel can be combined with interpolated diffuse and/or specular colour resulting in a colour sample of the surface with associated (generally non-integer) screen coordinates which indicate where this texture sample is mapped to on screen.

The screen space resampler splats mapped texels to integer screen positions, providing the image of the primitive on the screen. The 2D resampling operations can efficiently[2] be executed in two 1D resample passes using 1D FIR filter structures.

The pixel fragments coming from the screen space resampler are then combined in the EAA & HSR unit, which uses a fragment buffer[3]. Pixel fragments are depth-sorted into this buffer to solve the hidden surface problem. After all primitives have been rendered, all visible fragments for each pixel are combined (which mostly amounts to simple summation since the screen space resampler delivers colours already weighted by the prefilter) and sent to the frame buffer. Edge anti-aliasing results from combining the partial contributions generated by the screen space rasteriser near the edges, resulting in a final pixel colour which is a combination of colours from different primitives.

Techniques used in the A-buffer[1], the Z3 buffer[5] or the WarpEngine[14] could be used in the EAA & HSR unit, although the fragment buffer described in the paper from Meinds[3] differs from these in that it sees the colours in the buffer from one pixel as partial contributions to the whole filter footprint of that pixel (possibly partially behind one another) instead of colours for a certain position on a super-sample grid (which are always next to each other).

3. A combined architecture

Pixel shading often amounts to combining several textures in one way or another. This can easily be done in a traditional architecture since the texture space resampler resamples all textures to the common screen pixel grid, where the texture samples for each pixel can be combined.

In the forward texture mapping architecture, texture samples are only available on the common screen pixel grid after the screen space resampler stage. Also, a rasteriser can only traverse one grid at a time, so when rasterisation is taking place on a texture grid as in the forward mapping architecture and multi-texturing is enabled, the rasteriser has to select one texture out of many, and traverse the primitive on the associated grid. Multiple textures can be handled in a multi-pass fashion, so that they can be combined after they are resampled to the screen pixel grid. But that congests the fragment buffer in the EAA & HSR unit. It also precludes advanced features such as dependent texturing, or texture modes which are of an inverse nature, such as environment-mapped bump mapping (where the bump map information at each grid position determines where the environment map is indexed, possibly resulting in a one-to-many forward mapping from environment map to the primitive surface and the screen).

To avoid these problems we can make sure that the screen space resampler can map texture samples as if there was only one texture map associated with the primitive. To this end, shading of the surface colour should be performed *before* the screen space resampling process. This is depicted in Figure 3 which shows a hybrid graphics pipeline.

3.1. Overview

In this hybrid pipeline, the rasteriser traverses the primitive over a "surface grid", i.e. over a grid in a coordinate system that provides a 2D parameterisation of the surface of the primitive. The grid associated with a texture map provides such a surface grid, and is preferably used as surface grid (since obtaining texture samples on a texture grid does not require resampling). But in case of absence of texture maps, or when for example textures are 1D or 3D, another grid may be chosen. This is described in more details in Section 3.2. Attributes can be linearly interpolated over this grid, except (as in the case of forward texture mapping) for the perspectively mapped screen coordinates associated with each grid position.

The rasteriser interpolates attributes (including secondary texture coordinates) to the positions on this grid. The pixel shader then operates on the attributes on grid positions in this surface grid and if there are any secondary textures associated with the primitive, it uses inverse mapping with standard texture space resamplers to obtain colours from these

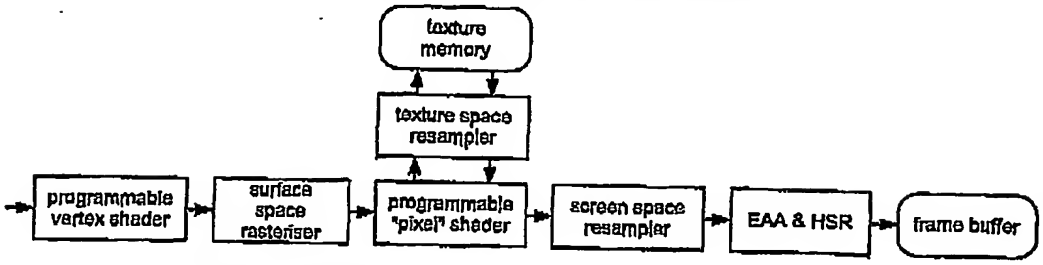


Figure 3: *Pixel shading and forward mapping architectures combined*

textures. If the surface grid was selected from a texture map, fetching texels for this "primary" texture stage boils down to the 4D mipmap reconstruction process from the forward mapping pipeline. This process is a form of axis-aligned anisotropic filtering, and anisotropic filtering is often available in standard texture space resamplers. The fetched texture samples and other attributes can be used by the pixel shader program to compute the surface colour for the current grid position.

Once the sample on the surface grid has been shaded, the screen space resampler is used to splat the colour to the screen pixels, where an EAA & HSR unit can combine contributions from different primitives.

3.2. Details

Using the hybrid architecture has several consequences and enables several nice features. These will be discussed here for the different stages of the hybrid pipeline along with some discussion of changes to these stages with respect to the traditional pipeline.

The rasteriser will have to choose a primitive grid (3.2.1), avoid bottleneck and shear problems (3.2.2), and control mipmapping (3.2.3). The programmable pixel shader and texture space resamplers remain the same as in a traditional pixel shading pipeline (3.2.4), and there are some options for the screen space resampler (3.2.5).

3.2.1. Choosing the surface grid

The surface space rasteriser will contain an extra first pipeline stage (next to the regular setup and traversal stages) in which the surface grid is chosen. This is done before the regular rasteriser setup.

Preferably, the surface grid is derived from one of the texture maps, so that this texture map does not have to be resampled (apart from 4D mipmapping) by the texture space rasteriser. To this end, the grid setup stage can examine the texture maps which are associated with the primitive. For a texture map to be eligible to provide the surface grid, it has to full-fill three requirements. First, it must not be addressed dependently. Second, it has to be a 2D texture (1D and 3D textures are not suitable for traversing a 2D surface). Third, the texture coordinates at the vertices should not make up a degenerate primitive (where for example all the texture coordinates line up, yielding in effect a 1D texture).

If more than one textures are eligible, we select the texture with the largest area in texture space: this is the texture with potentially the most detail and highest frequencies (so best to avoid the texture space resampling process for this texture, since this process can give rise to unneeded blur and aliasing).

If there is no eligible texture available (in case of a simple Gouraud shaded primitive for example), a dummy grid over the surface of the primitive can be chosen for the rasteriser to traverse, by assigning dummy "texture coordinates" to each vertex. This dummy grid is then traversed by the rasteriser as if it were a texture grid (except that texture fetching for these coordinates is disabled). An advantage is that the resulting linear interpolation over the surface provides for perspective correct Gouraud shading, as noted by Chen[3]. Assigning the x and y screen coordinates of each vertex as dummy texture coordinates is an option. Note that this choice does not mean that the rasteriser traverses screen pixels, since the homogeneous w for each vertex is still taken into account when mapping a 'texel' to the screen. Another option (for planar surfaces such as triangles) is rotating the 3D vertex coordinates in eye space, such that the normal of the rotated surface aligns with the eye space z-axis, and then selecting the rotated eye space x and y coordinates as dummy grid coordinates for each vertex.

The ability to traverse any grid over the surface of the primitive provides a lot of freedom. This freedom could be used for example to avoid any bottleneck and shear problems which might be associated with the mapping of a particular texture grid to the screen (see Section 3.2.2). It could be used to traverse non-planar primitives (such as traversing a Bezier patch via the 2D grid determined by its surface parameterisation, and using the forward mapping to directly splat the surface colours to the screen). It could also be used to rasterise in the direction of the motion of the primitive so that

a 1D filter can be applied along the rasterisation direction to efficiently implement motion blur[7].

3.2.2. Bottleneck and shear

If the screen space resampler is implemented using 2-pass filtering (see Section 3.2.5), the rasteriser has to prepare for this in addition to the setup required for primitive surface traversal. Two-pass filtering is known to suffer from bottleneck and shear problems[17, 15, 2].

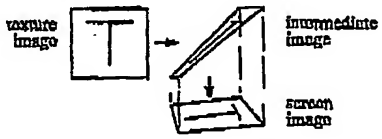


Figure 4: *The bottleneck problem*

The bottleneck problem is illustrated in Figure 4 where the area of the intermediate image becomes very small relative to the input and output images. It occurs with rotations close to 90°, and results in excessive blur in the direction of the second pass (since the second pass has to magnify the collapsed intermediate image again).

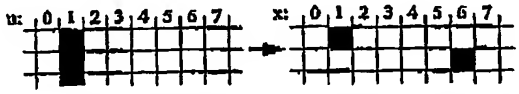


Figure 5: *Shear causing vertical aliasing in a horizontal pass*

The shear problem is illustrated in Figure 5 where two lines of a texture map and intermediate image are shown. The texture map has a black vertical line, and the shear of the perspective mapping is such that the black pixel on the second line ends up five pixels more to the right than the black pixel on the first line. The horizontal filter pass prevents horizontal aliasing, but not vertical aliasing. The shear causes a very sharp transition between black pixels on one line, and white pixels on the next. Also, the line in the intermediate image consists of disjunct parts, separated by columns to which the line does not contribute (e.g. for $x = 3$).

A solution[17] to the shear problem is to rasterise at a finer resolution, causing extra intermediate lines to be generated, with a black pixel at intermediate positions filling the holes. If this is done for first pass shear, the second pass will downscale the larger intermediate image to its final resolution. If shear in the second pass would be treated using the same super-sampling approach, a third pass would be needed to reduce the generated higher horizontal resolution to the output resolution.

Much like in the paper by Catmull and Smith[2], we avoid the bottleneck problem by choosing between the four options obtained by deciding (1) to generate the output image straight away, or generate a transposed version and transpose the generated image, and (2) doing the horizontal pass first, or doing the vertical pass first. However we use different criteria to choose between these options to try to avoid shear in the second pass. In this way, we will not have to generate extra image columns to counter shear in the second pass, but at worst have to only generate intermediate lines to counter shear aliasing in the first pass. This avoids having the penalty of a third pass.

Consider a local linearisation of the mapping around some point p:

$$\begin{pmatrix} x-x_p \\ y-y_p \end{pmatrix} = \begin{pmatrix} \frac{\partial x}{\partial u} & \frac{\partial x}{\partial v} \\ \frac{\partial y}{\partial u} & \frac{\partial y}{\partial v} \end{pmatrix} \begin{pmatrix} u-u_p \\ v-v_p \end{pmatrix}$$

The derivatives used there are good indicators for the bottleneck and shear problems (at point p): for the case where in the first pass u is mapped to x and in the second pass v to y, $\frac{\partial x}{\partial u}$ is the scale factor for the first pass: if this is small (close to zero), the intermediate image will collapse and exhibit the bottleneck problem, $\frac{\partial y}{\partial v}$ is the scale factor for the second pass, and $\frac{\partial x}{\partial v}$ and $\frac{\partial y}{\partial u}$ are the amounts of shear for the first and second pass respectively.

Each of the four derivatives is in the role of first pass scale factor for one of the four options for combating the bottleneck problem. Choosing the option which corresponds to the largest first pass scale factor (thereby maximising the intermediate image area[2]) reduces the bottleneck problem, but may leave shear in the second pass rather than in the first. To also take shear into consideration, we select between the four options in two stages.

First we decided how to pair up the coordinates: map u to x and v to y, or map u to y and v to x. We do this by looking at the derivatives at the primitives vertices and see how well u and v correlate to x and y respectively. We pair up the coordinates such that the two derivatives that correlate the strongest become scale factors, and the other two become shear factors. This amounts to the "transpose or not" choice from Catmull and Smith.

Second, we choose the order of the passes. Swapping the order of the passes puts the first pass scale and shear factors in the second pass, and vice versa. We choose the order that puts the least amount of shear in the second pass. This choice also determines which of the two scale factors will be the one for the first pass, and which one will be the scale factor for the second pass. This may yield a sub-optimal choice as far as the bottleneck problem is concerned, but since the first criterion selects strongly correlated coordinates as scale factors, both scale factors are usually large enough to avoid the bottleneck problem. So this selection process provides a good compromise between avoiding bottleneck and second pass shear problems.

3.2.3. Mipmapping

The rasteriser must sample the surface of the primitive at a resolution suitable for the resolution of its projected screen image. To this end, the grid is traversed in a way similar to mipmapping in a traditional pipeline. When more or less detail is required, the rasteriser can change mipmap level by shifting the delta values used to increment the grid coordinates and the interpolated attributes. In this way, the rasteriser takes larger or smaller steps over the surface grid.

The rasteriser maintains the screen coordinates associated to each grid position via the perspective mapping. Using these, it can determine if a mipmap switch is in order by making sure that the difference between the screen coordinates of subsequent grid positions remains within a suitable range (between $\frac{1}{2}$ and 1 for example, though a mipmap level bias can be employed of course).

The above-mentioned process does not directly provide the mipmap level to be used for fetching samples from any associated texture map, as it only ensures a proper surface-to-screen mipmap level. For each texture map, there is also a scaling factor associated with the mapping from the texture grid to the surface grid. This scaling factor corresponds to a texture-to-surface mipmap level which can be added to the surface-to-screen mipmap level to arrive at the required overall texture-to-screen mipmap level.

The texture-to-surface mipmap level can directly be obtained from the delta values used to interpolate the texture coordinates over the surface of the primitive, and since this interpolation is linear, the texture-to-surface mipmap level is constant per primitive and can be stored by the rasteriser setup in a register associated to each texture resampling stage for use as an offset.

3.2.4. Programmable shading and texture space resampling

The programmable "pixel" shader and texture space resampler (of which there may be one or more, for serial or parallel fetching of texture samples) are exactly the same as those in a traditional pipeline. The pixel shader receives a set of (interpolated) attributes, including texture and screen coordinates, for one location. The texture coordinates, along with the shading program, determine where to index the texture maps via the texture space resampler. The shader can also modify texture coordinates before sending them to the texture space resampler to implement dependent texturing, exactly in the same way as in a traditional pipeline.

The programmable shader passes the shaded colour on to the screen space resampler, along with the associated screen coordinates. These in general are not integer, but this is similar to how a pixel shader in a traditional pixel shader pipeline might receive sub-pixel screen positions when performing super-sampling . The shaded colour is the result of computations for one location, and does not depend on the grid traversed by the rasteriser. This means that existing shader programs will not have to be modified to run on the proposed architecture.

There are advantages and disadvantages to performing the programmable shading in surface space. Next to the high-quality anti-aliasing enabled by the forward mapping, a main advantage is a separation of concerns when it comes to the perspective mapping. The texture space resampler now does not have to deal with the perspective mapping to the screen. Most of the time it will be used to perform an affine transformation from texture grid to another. Standard bi-linear/tri-linear probe based texture space resamplers can better approximate the filter footprints required for such an affine mapping than the more general shaped footprints required for perspective mapping, so the quality from this resampling process will be higher. Only the screen space resampler has to deal with the perspective resampling, and it is only applied to shaded samples on the surface grid once.

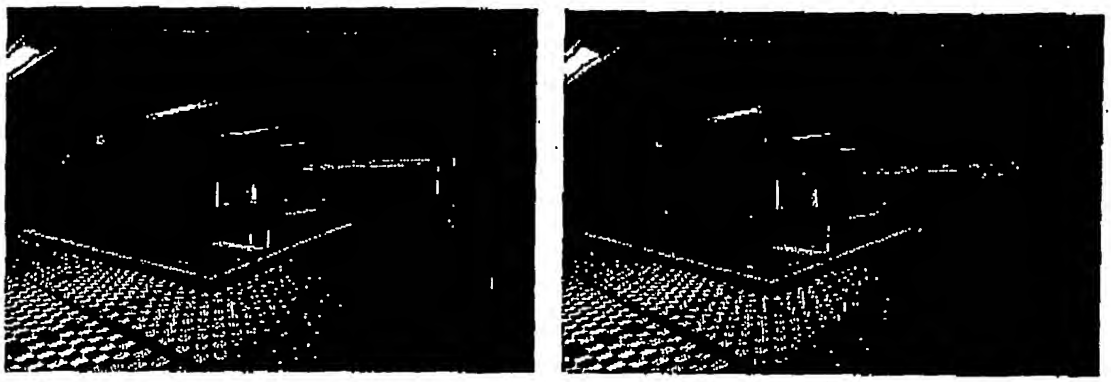
A disadvantage is that more samples are shaded than in a traditional pixel shading pipeline, since we have roughly twice as many surface samples as final pixels. This is due to mipmapping maintaining a minification factor between 1 and 2 in each direction (so we have roughly 1.5x1.5 surface samples per pixel). The high-quality anti-aliasing will however make sure that sub-pixel details still contribute to the final image, thereby further improving image quality.

Another disadvantage can be that secondary textures are now resampled twice (once by the texture space resampler and again by the screen space resampler), which might introduce extra blurriness. This is why we select the texture map with the highest detail as primary texture, to ensure that the finest details are only resampled once. The secondary textures will have a smaller minification (or even magnification, as is usually the case with for example light maps) for which some extra blur is not very noticeable. The screen space resampler also enables the use for high quality sharpness-enhancement filters, which can also help with maintaining a sharp image.

3.2.5. The screen space resampler

The shaded colour sample resulting from the pixel shading process is forwarded to the screen space resampler along with its screen coordinates. The screen space resampler resamples these colour samples, (located generally at non-integer pixel positions) to the integer pixel positions needed for display.

When rendering larger primitives or using a wide prefilter, a two-pass approach using 1D filters is more efficient than directly using a 2D filter kernel. Also, the 1D resamplers can be constructed such that no re-normalisation is required to avoid DC ripple[9]. Two-pass filters do require extra setup to alleviate for example the bottleneck and shear problems discussed in Section 3.2.2. The alternative is to use 2D resample kernels to perform the splatting.



a b

Figure 6: *A scene rendered with (a) a traditional pipeline using 2x2 multisampling, (b) the proposed hybrid pipeline*

The presence of this screen space splatting filter also enables direct support for point primitives[18], which widens the applicability of the pipeline.

The resulting pixel fragments are forwarded to the EAA & HSR unit which is the same as in the forward mapping pipeline discussed in Section 2.2.

4. Results

To demonstrate the concept, we took the software prototype[9] of the forward mapping pipeline and extended it, so the rasteriser can traverse arbitrary grids over the surface of the primitive. Since this pipeline was based on an older version of Mesa[10], we did not have a full pixel shader available, so we decided to only implement multi-texturing to show that the concept works.

The degree of programmability of the shader is relatively unimportant when showing that it is easy to combine several textures, so all that is lacking from this prototype is a demonstration of dependent texturing, but since the pixel shader is the same as in a traditional pipeline, exactly the same modifications to the texture addresses can be made prior to fetching texels, so the support for dependent texturing is trivial.

Figure 6 shows a comparison of a scene[16] from the game Quake III rendered by a traditional pipeline and the prototype of the hybrid pipeline. Picture 6a exhibits aliasing for example on the staircase, even though 2x2 multi-sampling was enabled. In picture 6b, this aliasing is virtually gone, even though the computational and bandwidth costs for generating this picture are roughly the same as for picture 6a. The shadows on the floor and lamp-light on the walls attest to the use of light-maps, which are rendered using the multi-texturing "shader" of the prototype of the hybrid pipeline.

5. Conclusions

A traditional inverse mapping pipeline can be transformed into the proposed hybrid inverse/forward mapping pipeline. This involves generalising the rasteriser to enable rasterisation over a surface grid. It also involves adding a screen space resampler. And it involves exchanging the multi-sample buffer, z-buffer and down-sample filter in the HSR & EAA unit for a fragment buffer and fragment combiner logic.

The combined architecture features both the advantages offered by programmable pixel shading and the high quality anti-aliasing offered at low cost by the forward mapping technique. Existing pixel shading programs can execute on the new architecture without modification.

The proposed architecture also enables new features, such as support for point primitives (due to the addition of the screen space resampler which performs the required splatting), and support for motion blur (due to the ability of the rasteriser to traverse the primitive in the direction of motion).

References

1. Loren Carpenter. The A-buffer, an antialiased hidden surface method. In Hank Christiansen, editor, *Computer Graphics (SIGGRAPH '84 Proceedings)*, volume 18, pages 103–108, July 1984.

2. Edwin Catmull and Alvy Ray Smith. 3-d transformations of images in scanline order. In *Computer Graphics (SIGGRAPH '80 Proceedings)*, volume 14(3), pages 279–285, 1980.

3. Baoquan Chen, Frank Dachille, and Arie Kaufman. Forward image mapping. In *Proceedings of IEEE Visualization '99*, pages 89–96, October 1999.

4. Paul S. Heckbert. Survey of texture mapping. *IEEE Computer Graphics and Applications*, Nov. 1986.

5. Norman P. Jouppi and Chun-Fa Chang. Z3: An economical hardware technique for high-quality antialiasing and transparency. In *Proceedings of Eurographics/Siggraph workshop on graphics hardware 1999*, pages 85 – 93. ACM Press, 1999.

6. Erick Lindholm, Mark J. Kilgard, and Henry Moreton. A user-programmable vertex engine. In *Proceedings Siggraph 2001*, pages 149 – 158, August 2001.

7. Koen Meinds. Temporal anti-aliasing with forward texture mapping. In *Submitted to Graphics Hardware 2003*, 2003.

8. Koen Meinds and Bart Barenbrug. Resample hardware for 3d graphics. In T. Ertl, W. Heidrich, and M Doggett, editors, *Proceedings of Graphics Hardware 2002*, pages 17 – 26, 2002.

9. The *resample hardware for 3d graphics* webpage at http://www.extra.research.philips.com/graphics/, 2003.

10. The Mesa homepage at http://www.mesa3d.org/.

11. Microsoft. *DirectX Graphics Programmers Guide*. Microsoft Developers Network Library.

12. M. Olano, J.C. Hart, W. Heidrich, and M. McCool. *Real-Time Shading*. A K Peters, Natick, Massachusetts, 2002.

13. M. Olano and A. Lastra. A shading language on graphics hardware: The pixelflow shading system. In *Proceedings Siggraph 1998*, pages 159 – 168, July 1998.

14. Voicu Popescu, John Eyles, Anselmo Lastra, Joshua Steinhorst, Nick England, and Lars Nyland. The warpengine: An architecture for the post-polygonal age. In *Proceedings Siggraph 2000*, pages 433 – 442, 2000.

15. Alvy Ray Smith. Planar 2-pass texture mapping and warping. In *Computer Graphics (Proceedings Siggraph 1987)*, volume 21(4), pages 263 – 272, 1987.

16. Quake III level "Subversive Tendencies" available at http://bellenberg.home.mindspring.com/tequ3rny3.html by Tequila, which uses textures maps from http://www.planetquake.com/hfx/ some of which we replaced, and for some of which we increased the contrast.

17. George Wolberg and Terrance E. Boult. Separable image warping with spatial lookup tables. In *Computer Graphics (Proc. Siggraph '89)*, volume 23(3), pages 369 – 378, July 1989.

18. Matthias Zwicker, Hanspeter Pfister, Jeroen van Baar, and Markus Gross. Surface splatting. In *Proceedings Siggraph 2001*, pages 371 – 378, August 2001.

This Page is inserted by IFW Indexing and Scanning Operations and is not part of the Official Record

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- [x] BLACK BORDERS
- [x] IMAGE CUT OFF AT TOP, BOTTOM OR SIDES
- [x] FADED TEXT OR DRAWING
- [ ] BLURED OR ILLEGIBLE TEXT OR DRAWING
- [ ] SKEWED/SLANTED IMAGES
- [x] COLORED OR BLACK AND WHITE PHOTOGRAPHS
- [ ] GRAY SCALE DOCUMENTS
- [ ] LINES OR MARKS ON ORIGINAL DOCUMENT
- [ ] REPERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY
- [ ] OTHER: _____

IMAGES ARE BEST AVAILABLE COPY.
As rescanning documents *will not* correct images problems checked, please do not report the problems to the IFW Image Problem Mailbox